Designing and Implementing a Family of Intrusion Detection Systems

Richard A. Kemmerer
Reliable Software Group
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106
USA

kemm@cs.ucsb.edu

Intrusion detection systems (IDSs) analyze information about the activities performed in a computer system or network, looking for evidence of malicious behavior. Attacks against a system manifest themselves in terms of events. These events can be of a different nature and level of granularity. For example, they may be represented by network packets, operating system calls, audit records produced by the operating system auditing facilities, or log messages produced by applications. The goal of intrusion detection systems is to analyze one or more event streams and identify manifestations of attacks.

The intrusion detection community has developed a number of different tools that perform intrusion detection in particular domains (e.g., hosts or networks), in specific environments (e.g., Windows NT or Solaris), and at different levels of abstraction (e.g., kernel-level tools and alert correlation systems). These tools suffer from two main limitations: they are developed *ad hoc* for certain types of domains and/or environments, and they are difficult to configure, extend, and control remotely.

In the specific case of signature-based intrusion detection systems the sensors are equipped with a number of attack models that are matched against a stream of incoming events. The attack models are described using an *ad hoc*, domain-specific language (e.g., N-code, which is the language used by the Network Flight Recorder intrusion detection system). Therefore, performing intrusion detection in a new environment requires the development of both a new system and a new attack modeling language. As intrusion detection is applied to new and previously unforeseen domains, this approach results in increased development effort.

Today's network are not only heterogeneous, but also dynamic. Therefore, intrusion detection systems need to support mechanisms to dynamically change their configuration as the security state of the protected system evolves. Most existing intrusion detection systems are initialized with a set of signatures at startup time. Updating the signature set requires stopping the IDS, adding new signatures, and then restarting execution. Some of these systems provide a way to enable/disable some of the available signatures, but few systems allow for the dynamic inclusion of new signatures at execution time. In addition, the *ad hoc* nature of existing IDSs does not allow one to dynamically configure a running sensor so that a new event stream can be used as input for the security analysis.

Another limitation of existing IDSs is the relatively static configuration of responses. Normally it is possible to choose only from a specific subset of possible responses. In addition, to our knowledge, none of the systems allows one to associate a response with *intermediate* steps of an attack. This is a severe limitation, especially in the case of distributed attacks carried out over a long time span.

Finally, the configuration of existing IDSs is mostly performed manually and at a very low level. This task is particularly error-prone, especially if the intrusion detection systems are deployed across a very heterogeneous environment and with very different configurations.

This talk describes a framework for the development of intrusion detection systems, called STAT, that overcomes these limitations. The STAT framework includes a domain-independent attack modeling language and a domain-independent event processing analysis engine. The framework can be extended in a well-defined way to match new domains, new event sources, and new responses. The resulting set of applications is a software family whose members share a number of features, including dynamic reconfigurability and a fine-grained control over a wide range of characteristics. The main advantage of this approach is the limited development effort and the increased reuse that result from using an object-oriented framework and a component-based approach.

STAT is both unique and novel. First, STAT is the only known framework-based approach to the development of intrusion detection systems. Second, even though the use of frameworks to develop families of systems is a well-known approach, the STAT framework is novel in the fact that the framework extension process includes, as a by-product, the generation of an attack modeling language closely tailored to the target environment. This talk focuses primarily on the STAT framework.

Paper presented at the RTO IST Symposium on "Adaptive Defence in Unclassified Networks", held in Toulouse, France, 19 - 20 April 2004, and published in RTO-MP-IST-041.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 NOV 2004	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Designing and Implementing a Family of Intrusion Detection Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Reliable Software Group Department of Computer Science University of California, Santa Barbara Santa Barbara, CA 93106 USA**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001845, Adaptive Defence in Unclassified Networks (La defense adaptative pour les reseaux non classifies)., The original document contains color images.

14. ABSTRACT

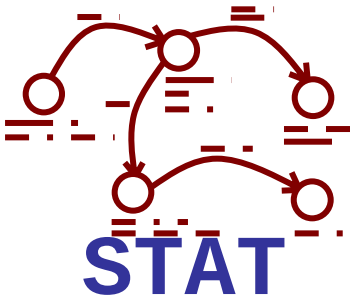
15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**56**	

Designing and Implementing A Family of Intrusion Detection Systems

Richard A. Kemmerer

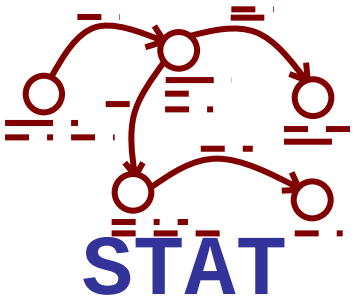
Reliable Software Group
Computer Science Department
University of California
Santa Barbara, CA 93106, USA
http://www.cs.ucsb.edu/~rsg/STAT/



Intrusion Detection

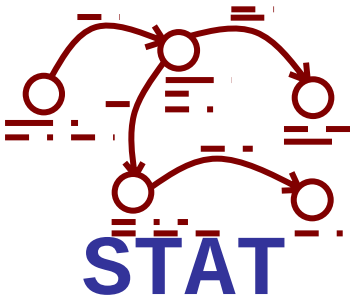
- Analysis of the actions performed by users and applications looking for evidence of malicious activities
- Two techniques
  - *Anomaly detection* (statistics, profiles, specs) (*IDES, RST, ADAM*)
    - Detects previously unknown attacks
    - Difficult to configure (train), generates many false alarms
  - *Misuse detection* (signature analysis) (*NFR, Emerald, Snort, STAT*)
    - Generates few false alarms
    - Detects only known attacks, needs continuous updating
- Different domains
  - Host-based
  - Application-based
  - Network-based

Undesirable Format for an Intrusion Report



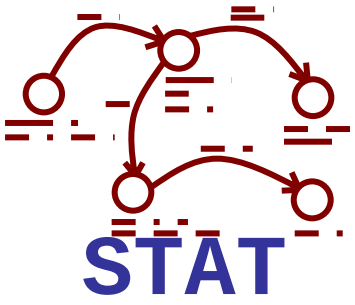
Intrusion Detection

- Intrusion detection is traditionally based on analysis of low-level events: network packets, system calls, audit records
- Intrusion detection has evolved in several ways
  - New analysis techniques
  - Multiple event sources, possibly introducing distribution
  - Abstraction: fusion/correlation of high-level events, e.g., alerts
- Monitor and surveillance functionality always/still based on sensors



Intrusion Detection Sensor Limitations

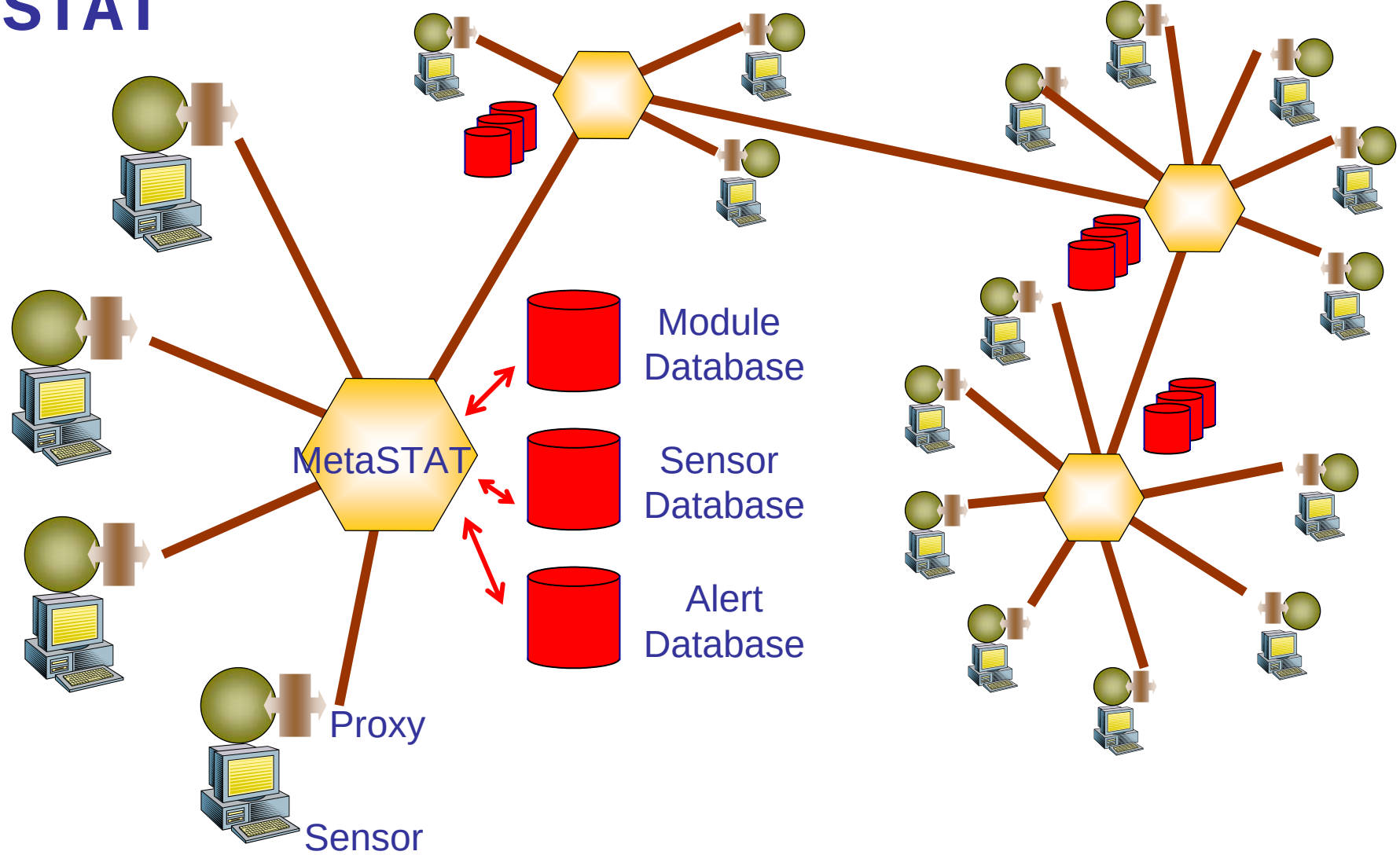
- Sensors are developed in an ad hoc fashion to match specific environments/domains/event sources
- Sensors are hard to configure
- Sensors are hard to control
- Sensors are hard to extend
- Configuration/control/extension is mostly executed statically
- Configuration is mostly done manually
- Identifying “meaningful” sensor configurations can be difficult
- Number of sensors that can be easily managed is small

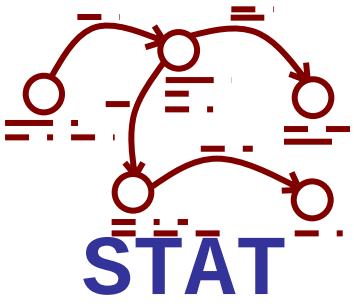


Solution: A Web of Sensors

- Set of heterogeneous sensors that provide intrusion detection functionality within a protected network
  - STAT Framework
  - STATL and the STAT core
- Sensors controlled, coordinated, and configured by means of a distributed infrastructure
  - MetaSTAT
- Explicit modeling of component dependencies and current sensor configuration supports automated “meaningful” reconfigurations

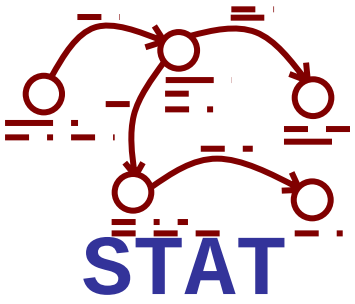
A Web Of Sensors





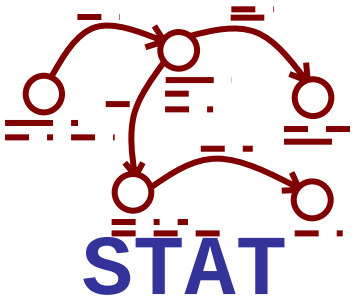
Outline

- STAT and STAT Framework
- STAT Extension Process
- MetaSTAT



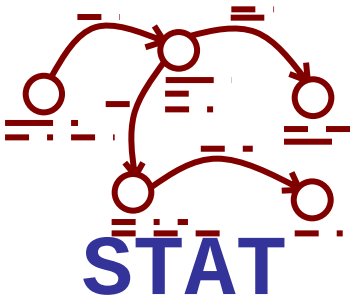
The STAT Framework

- Object-Oriented framework for the development of intrusion detection sensors
- Based on the State Transition Analysis Technique
- Provides a domain-independent attack modeling language, called STATL
- Provides a “core” domain-independent event processing analysis engine that implements the STATL semantics
- Supports a well-defined extension process
- Supports flexible and dynamic response mechanisms
- Provides a communication and control infrastructure



Set of IDS Applications is a Software Family

- A number of STAT-based sensors have been developed leveraging the framework
- The result is a “software family” whose members share a number of features
  - Dynamic reconfiguration
  - Fine-grained control (response, scenarios)
  - Attack specification language
- Limited development effort
- High level of reuse

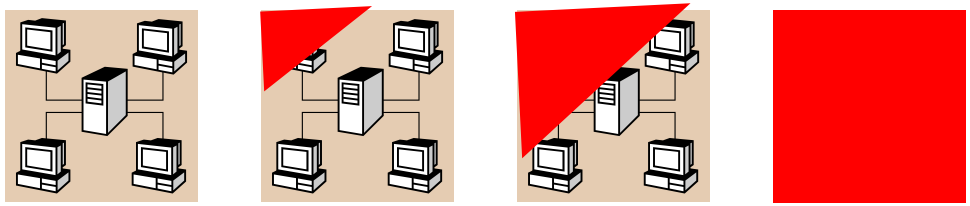


State Transition Analysis Technique

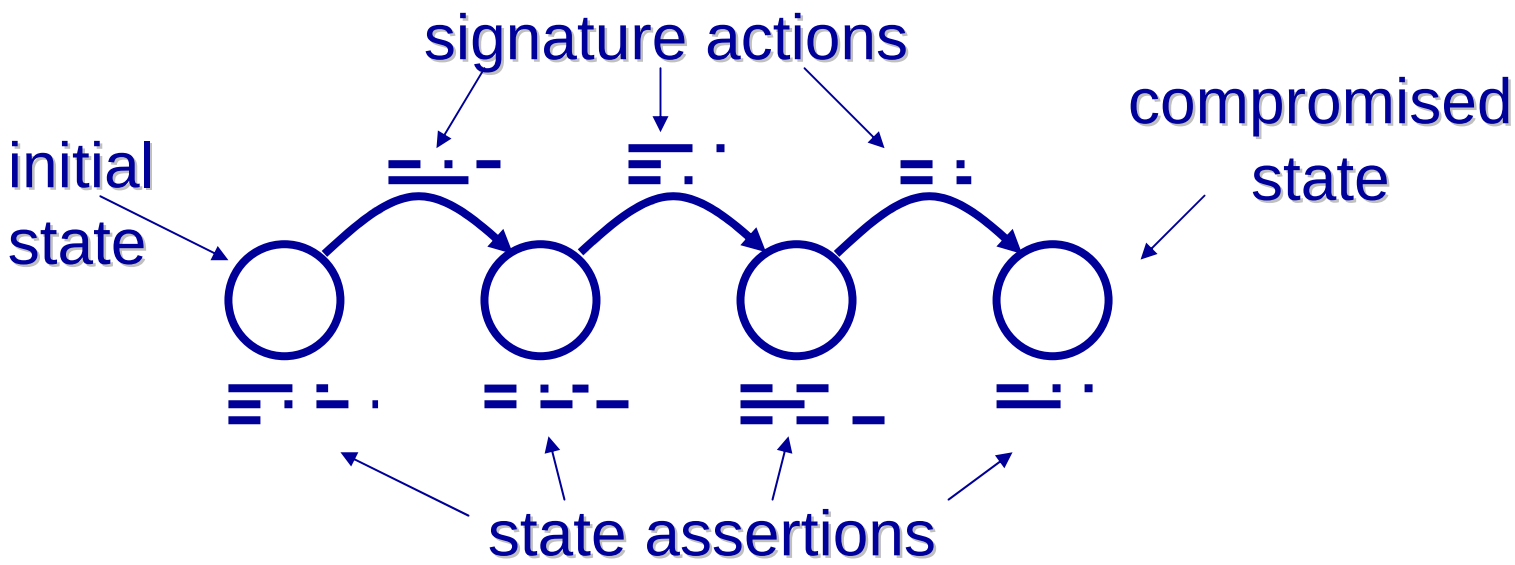
- STAT models penetrations as a sequence of state transitions
- Represents only key activities that lead from an initial safe state to a final compromised state
  - Signature Actions
  - State Assertions

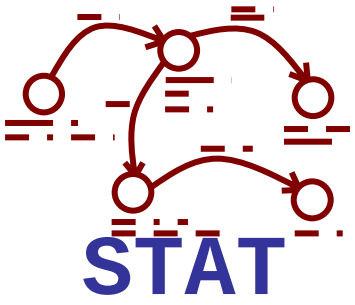
State Transition Diagrams

Attacker has limited privileges



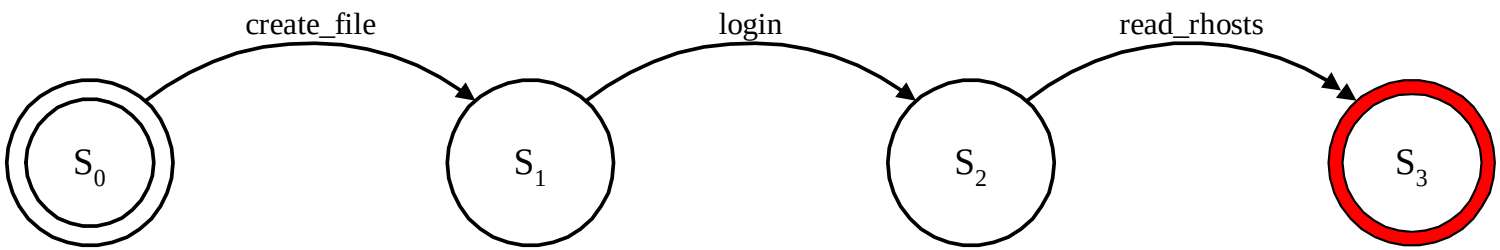
Attacker illicitly gains more privileges

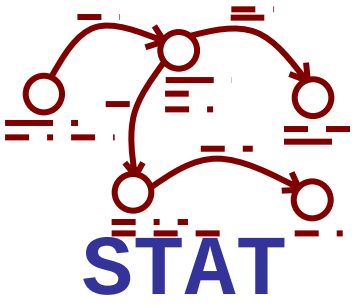




USTAT example ftp-write

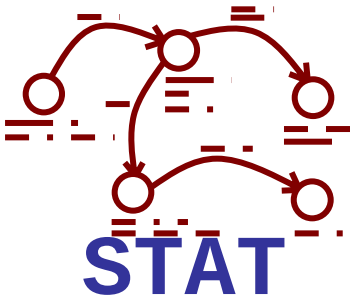
- Exploit
  - use ftp to create .rhosts file in world-writable ftp home directory
  - rlogin using bogus .rhosts file





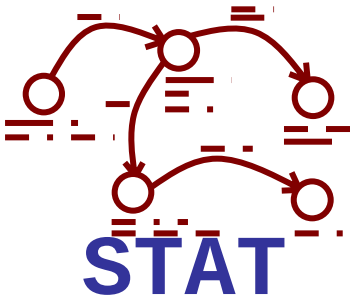
STATL

- A STATL specification is the description of a complete attack scenario (a signature) in terms of states and transitions
- Domain-independent language
  - Extensions for
    - IP networks
    - Solaris BSM
    - WinNT event logging facility
    - Apache event logs
    - Syslog facility
    - IDMEF Alerts
- Parameterized descriptions
  - Generic attacks customizable by installation or policy



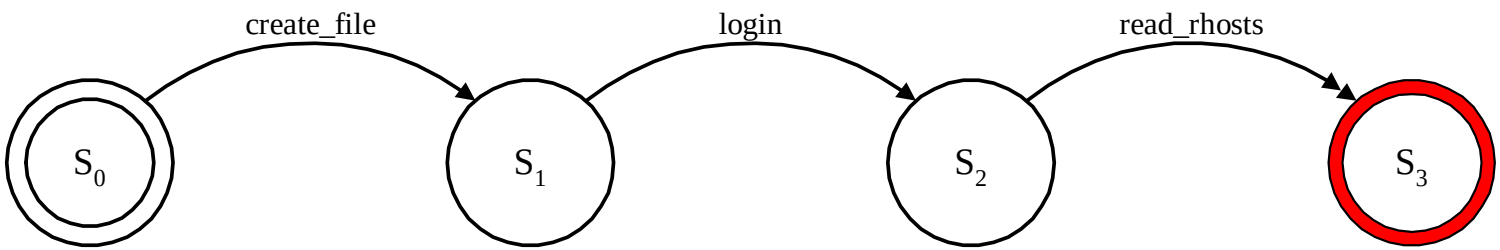
STATL Basic Abstractions

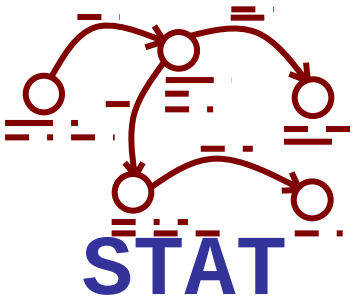
- Scenarios
  - States
  - Transitions (consuming, nonconsuming, unwinding)
  - Signature actions
  - Assertions
  - Global environment
  - Local environment
  - Code blocks
- Events
  - Defined as trees of generic events encapsulating domain-specific opaque events
- Timers



USTAT example ftp-write

- Exploit
  - use ftp to create .rhosts file in world-writable ftp home directory
  - rlogin using bogus .rhosts file

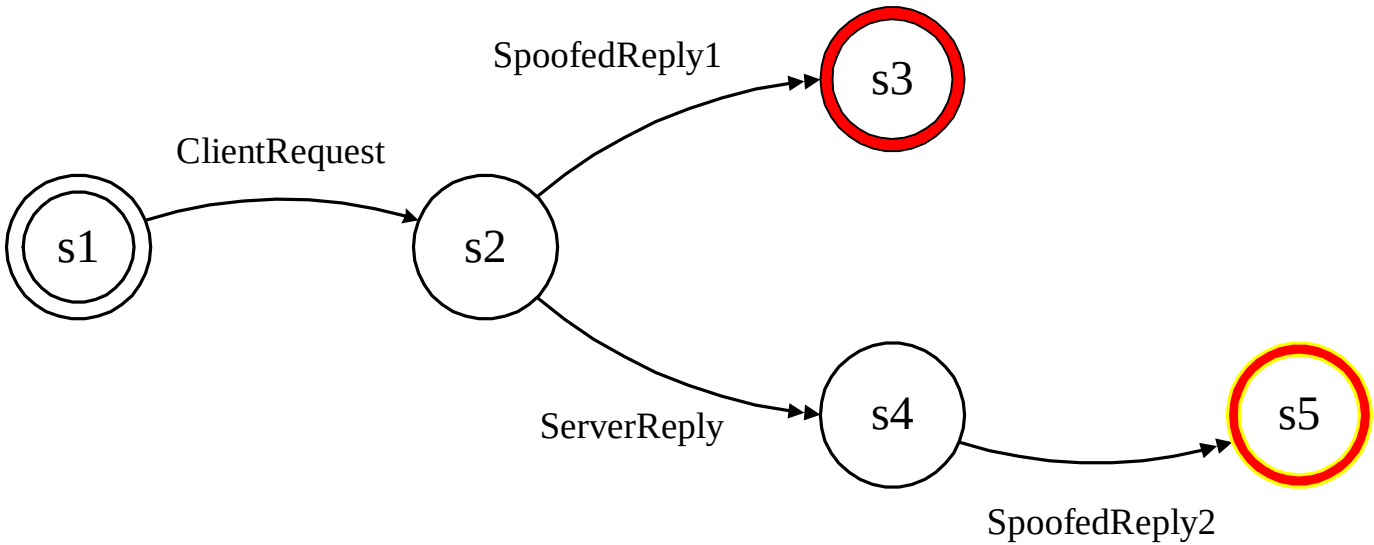


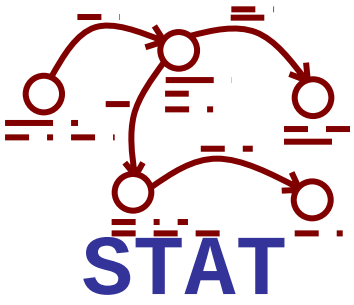


ftp-write in STATL

```
use ustat;

scenario ftp_write
{
   int user, pid, inode;
   string objname;

   initial state s0 { }

   transition create_file (s0 -> s1) nonconsuming
   {
      [WRITE w] : (w.euid != 0) && (w.owner != w.ruid)
      {
         inode = w.inode;
         objname = w.objname;
      }
   }

   state s1 { }

   transition login (s1 -> s2) nonconsuming
   {
      [EXECUTE e] : match_name(e.objname, "login")
      {
         user = e.ruid;
         pid = e.pid;
      }
   }

   state s2 { }

   transition read_rhosts (s2 -> s3) consuming
   {
      [READ r] : (r.pid == pid) && (r.inode == inode)
   }

   state s3
   {
      {
         string username = userid2name(user);
         log("%d: by user %s using %s", user, username, objname);
      }
   }
}
```

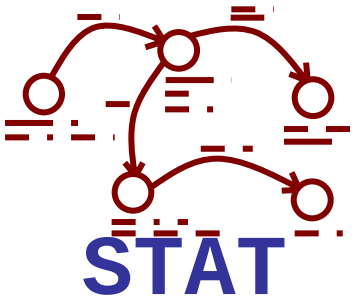
NetSTAT example UDP-race





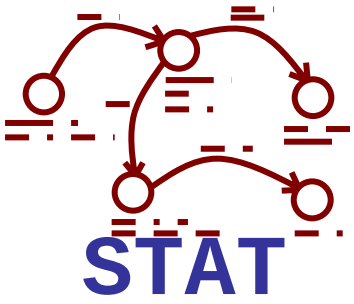
UDP-race in STATL

```
use netstat;
scenario UDP_race
{
   Host server, racer;
   Service x;
   IPAddress a_s, a_c;
   Interface i_r;

   IPAddress request_ip_src, request_ip_dst;
   Port request_udp_src, request_udp_dst;

   <* CONSTRAINT
     ( server in Network.hosts) &&
      (x in server.services) &&
      (x.protocol == "UDP") &&
      (x.authentication == "IPaddress") &&
      (a_s in x.ipAddresses) &&
      (a_c in x.trustedAddr) &&
      (a_c.interface in ProtectedNetwork.interfaces) &&
      (racer in Network.hosts) &&
      (racer != server) &&
      ! (a_c in racer.ipAddresses) &&
      i_r in racer.interfaces
   *>

   state S3 { { log("compromised"); } }

   state S5 { { log("under attack"); } }

   transition ClientRequest (S1 -> S2) nonconsuming
   {
      [IPDatagram d1 [UDPDatagram u1]]
        <* ENDPOINT_PORTS a_c.interface, a_s.interface *> :
          (d1.src == a_c) && (d1.dst == a_s) && (u1.dst == x.port)
      {
         request_ip_src = d1.src;
         request_ip_dst = d1.dst;
         request_udp_src = u1.src;
         request_udp_dst = u1.dst;
      }
   }

   transition ServerReply (S2 -> S4) consuming
   {
      [IPDatagram d2 [UDPDatagram u2]]
        <* ENDPOINT_PORTS a_s.interface, a_c.interface *> :
          (d2.src == request_ip_dst) &&
          (d2.dst == request_ip_src) &&
          (u2.src == request_udp_dst) &&
          (u2.dst == request_udp_src)
   }

   action SpoofedReply
   {
      [Message m2 [IPDatagram d2 [UDPDatagram u2]]]
        <* ENDPOINT_PORTS i_r, a_c.interface *>
        <* CONSTRAINT ! exists_detached_path(m2.src, d2.src.interface) *> :
           (d2.src == request_ip_dst) &&
           (d2.dst == request_ip_src) &&
           (u2.dst == request_udp_src) &&
           (u2.src == request_udp_dst)
   }

   transition SpoofedReply1 (S2 -> S3) consuming { SpoofedReply }

   transition SpoofedReply2 (S4 -> S5) consuming { SpoofedReply }
}
```



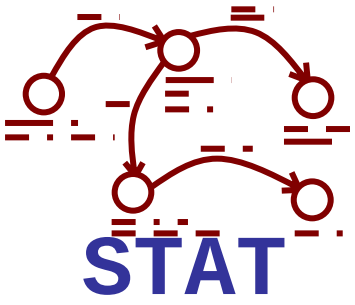
STATL Execution Model

- A STATL scenario has a runtime representation in terms of
  - Prototype (global environment and STD definition)
  - Instances (local environment, occurrence of an attack)
- Event matching and assertions determine which enabled transitions fire
- Scenario evolution determined by transition type
  - *Nonconsuming*: New instance in new state *and* current instance stays in previous state
  - *Consuming*: Current instance changes its state
  - *Unwinding*: Backtracking to ancestor instance, possibly removing a subtree of the instance tree



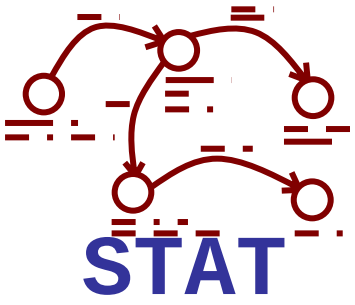
The STAT Core Module

- Implements STATL basic abstractions
  - Scenario
    - State
    - Transitions (consuming, non-consuming, unwinding)
    - Signature actions
    - Assertions
    - Global environment
    - Local environment
    - Code fragments
  - Events
  - Timers
  - Synthetic events
- Defines general semantics
  - Event matching
  - Scenario processing
  - Unwinding



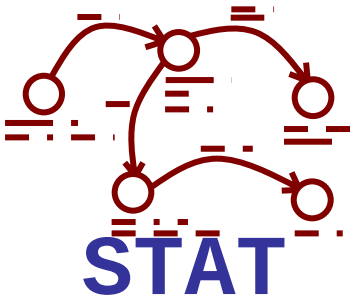
STAT Extension Process

- STATL language and the core analysis engine are both extended to deal with a specific domain (host, network, application), event stream, or platform
- Can be dynamically extended to build a STAT-based sensor
  - Language extensions
  - Event providers
  - Scenario plugins
  - Responses modules
- Extensions contain data structures and code to operate on the data structures



STAT Extension Process Uses a Common Format

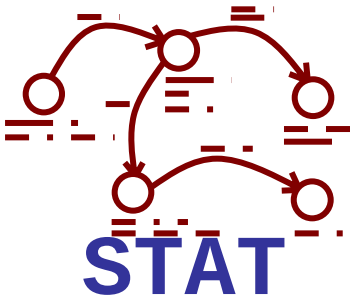
- A common extension format:
  - Saves a considerable amount of development time
  - Produces more reliable libraries
  - Allows for interchangeable event producers/consumers
- Uses C++ class hierarchy
  - Create, destroy, clone, dump, restore, type management
- Subclass STAT framework C++ root classes:
  - STAT_Event
  - STAT_Type
  - STAT_Provider
  - STAT_Scenario
  - STAT_Response



Language Extension

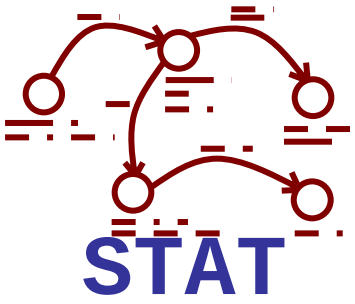
- Set of events and types that characterize the entities of a particular domain
- All event types defined as subclasses of STAT_Event
- All other types defined as subclasses of STAT_Type
- Defined events and types can be used in writing STATL scenarios for the specific domain

- Compiled into dynamically-linked libraries (.so or DLL files)
- Loaded into the core whenever needed by a scenario



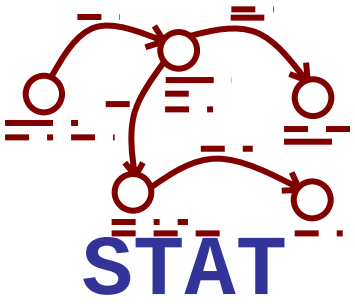
Attack Scenarios

- Written in STATL with the relevant language extensions
- Automatically translated into a subclass of the STAT_Scenario class
- Compiled into dynamically-linked libraries, called scenario plugins
- Loaded into the core as needed



Event Providers

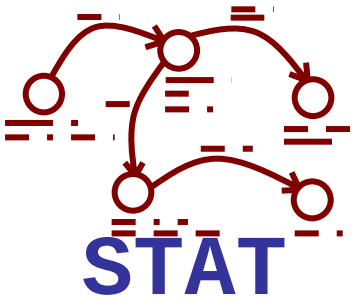
- Collect events from the environment
- Create STAT events as defined in one or more language extensions
- Insert the events in the event queue of the STAT core
- Created by subclassing the STAT_Provider class
- Multi-threaded runtime supports the processing of multiple event streams



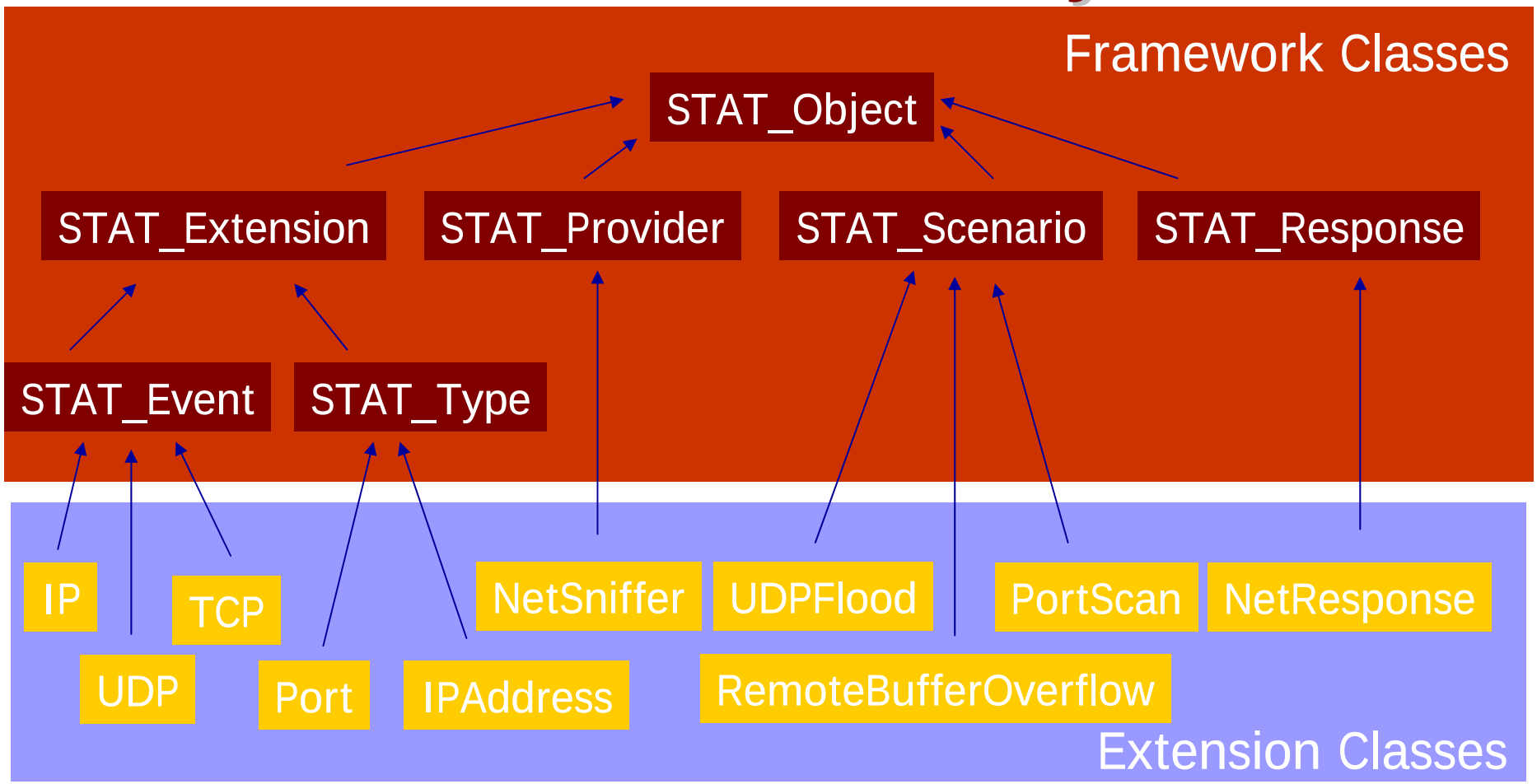
Response Modules

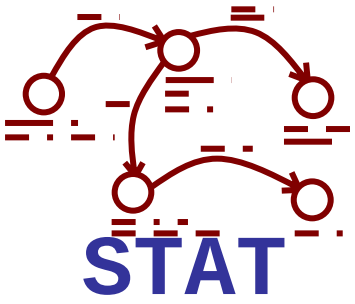
- Contain libraries of actions that may be associated with the evolution of one or more scenarios
- Created by subclassing the STAT_Response class
- Compiled into dynamically-linked libraries
- Loaded into STAT core when needed
- One or more actions can be associated with any state defined in a loaded scenario plugin

- Example actions
  - write to file
  - reset a TCP connection
  - email to Network Security Officer



STAT Framework Class Hierarchy

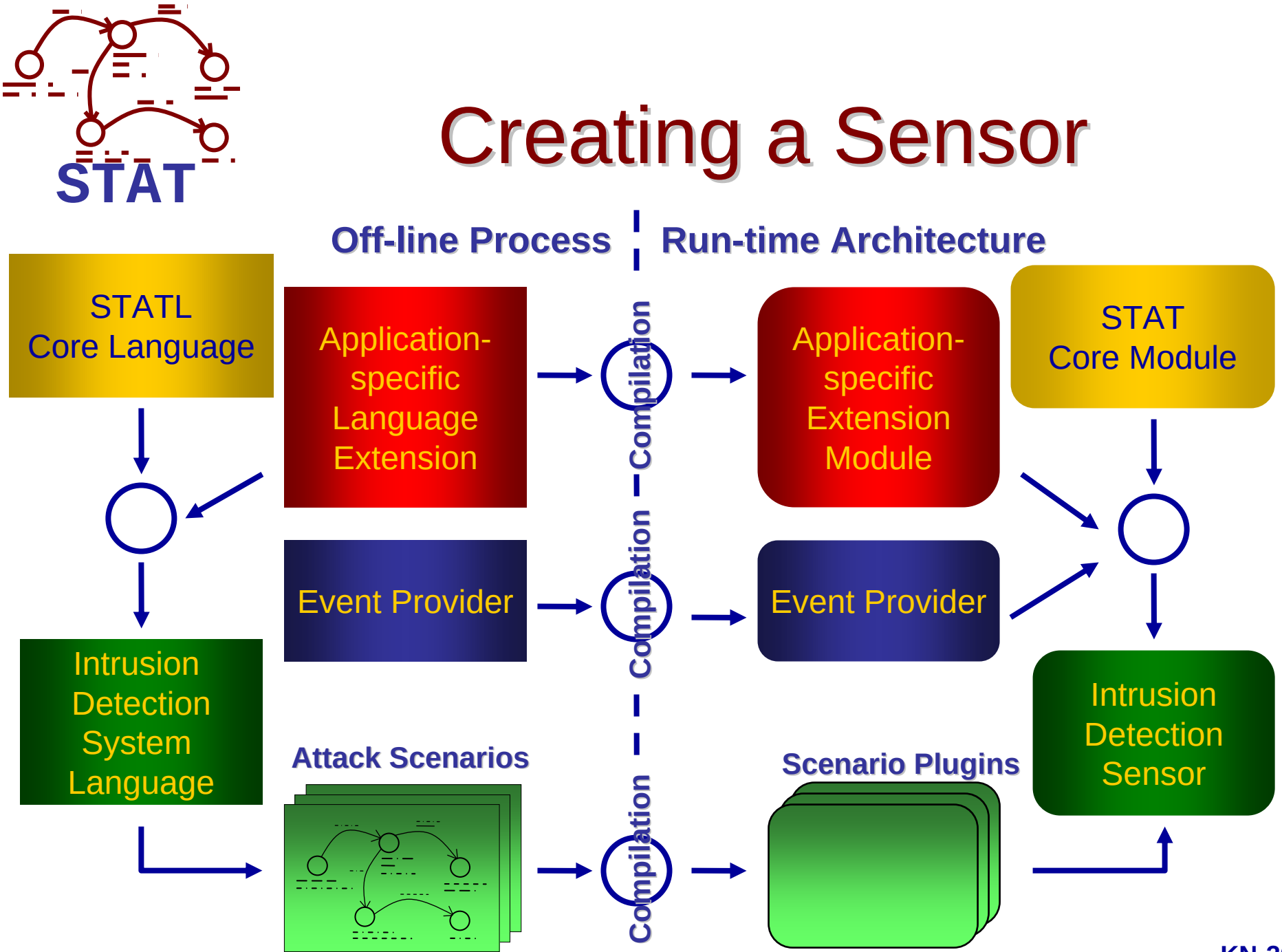


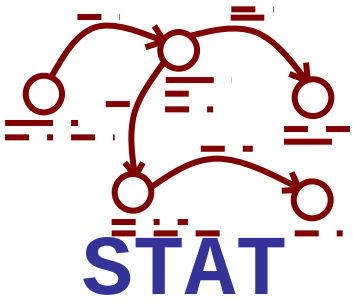


The Framework At Work

- Define a Language Extension, i.e., the events, types, and predicates to be used in a specific domain
- Compile the extension into a Language Extension Module
- Develop an Event Provider that transforms external data into events as defined by one or more Language Extensions
- Compile the Event Provider into a dynamically linkable module
- Develop STATL scenarios that use the events defined in one or more Language Extensions
- Translate/compile the scenario into a Scenario Plugin
- If necessary, develop response libraries to be used with the scenario
- Link everything together (shake well) and run your sensor

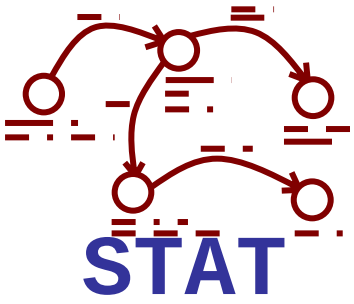
Creating a Sensor





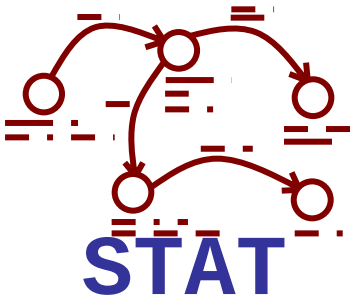
A Family Of Sensors

- USTAT (Host-based, Solaris, BSM auditing)
- NetSTAT (Network-based, Linux/Solaris, network traffic)
- WinSTAT (Host-based, Windows 2000, Security event logs)
- LinSTAT (Host-based, Linux platform, Snare auditing)
- WebSTAT (Application-based, UNIX, Apache logs)
- AlertSTAT (Correlator, UNIX, IDMEF alerts)



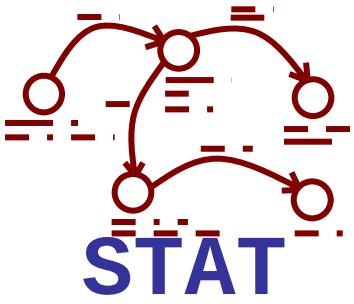
A Family Of Sensors

- LogSTAT (Host-based, UNIX OS, syslog files)
- ftpSTAT (Application-based, extension of LogSTAT)
- aodvSTAT (Network-based, Linux, Wireless Ad-Hoc routing protocol events)
- AgletSTAT (Application-based, Linux, Aglets mobile code system)
- SnortSTAT (Application-based, UNIX, Snort plugin)
- SienaSTAT (WAN Correlator, UNIX, SIENA events)



OK, You Can Develop Your Own IDS, But...

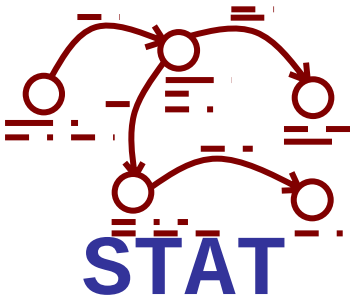
- What if one wants to change the configuration of a sensor at run time, without having to stop the whole thing?
- How can one be sure that all the pieces (extensions, providers, scenarios) fit together?
- What if one wants to control a multitude of sensors deployed throughout the network?
- What if one wants to aggregate/fuse/correlate the alerts produced by the deployed sensors?



MetaSTAT

A communication and control infrastructure for STAT-based sensors

- CommSTAT communication infrastructure allows for the exchange of alerts and control commands over secure connections
- MetaSTAT Controller dispatches commands to the sensors
- The STAT Proxy mediates communication
  - Performs local module management (installation/configuration)
  - Relays commands to sensors (loading/activation)

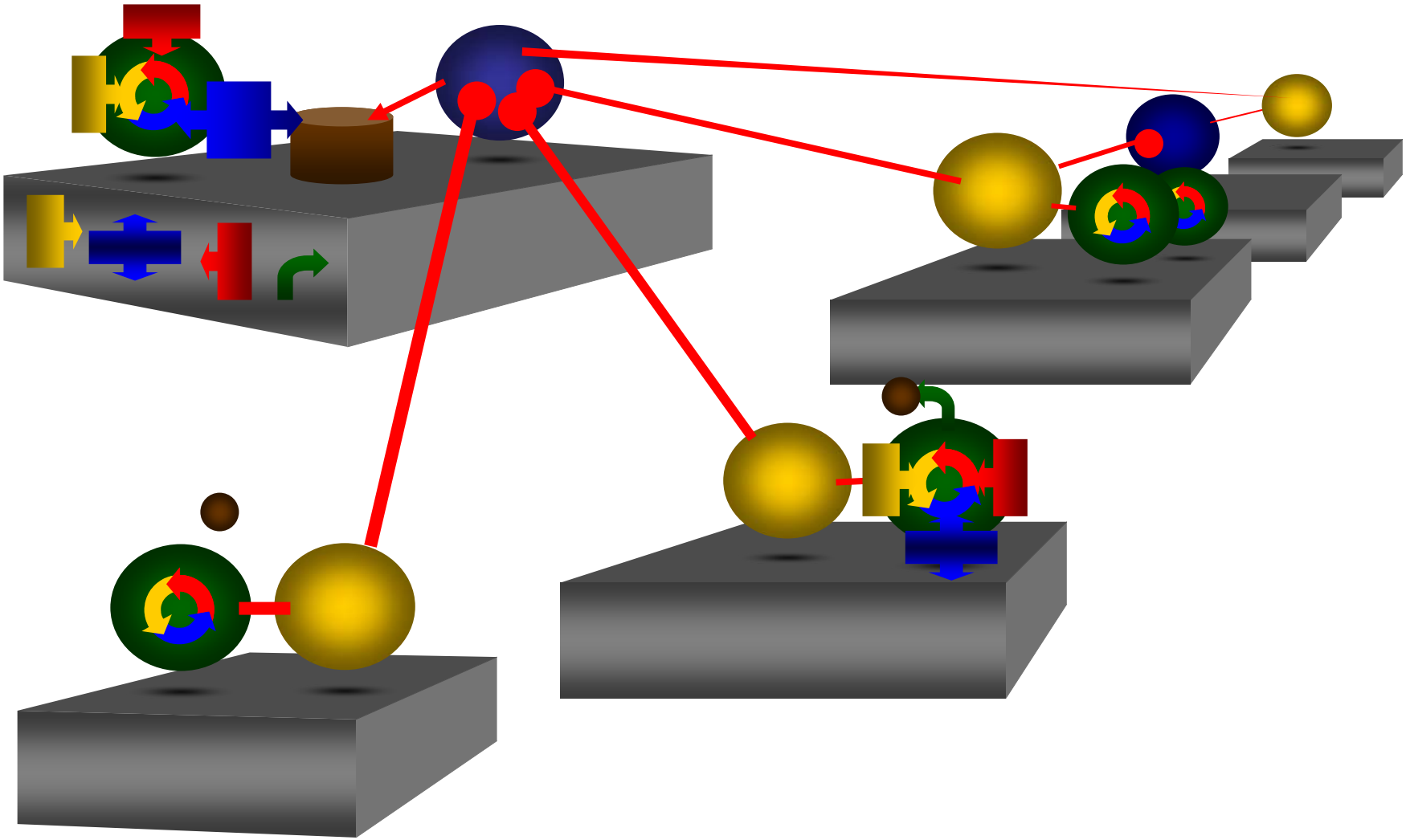


MetaSTAT

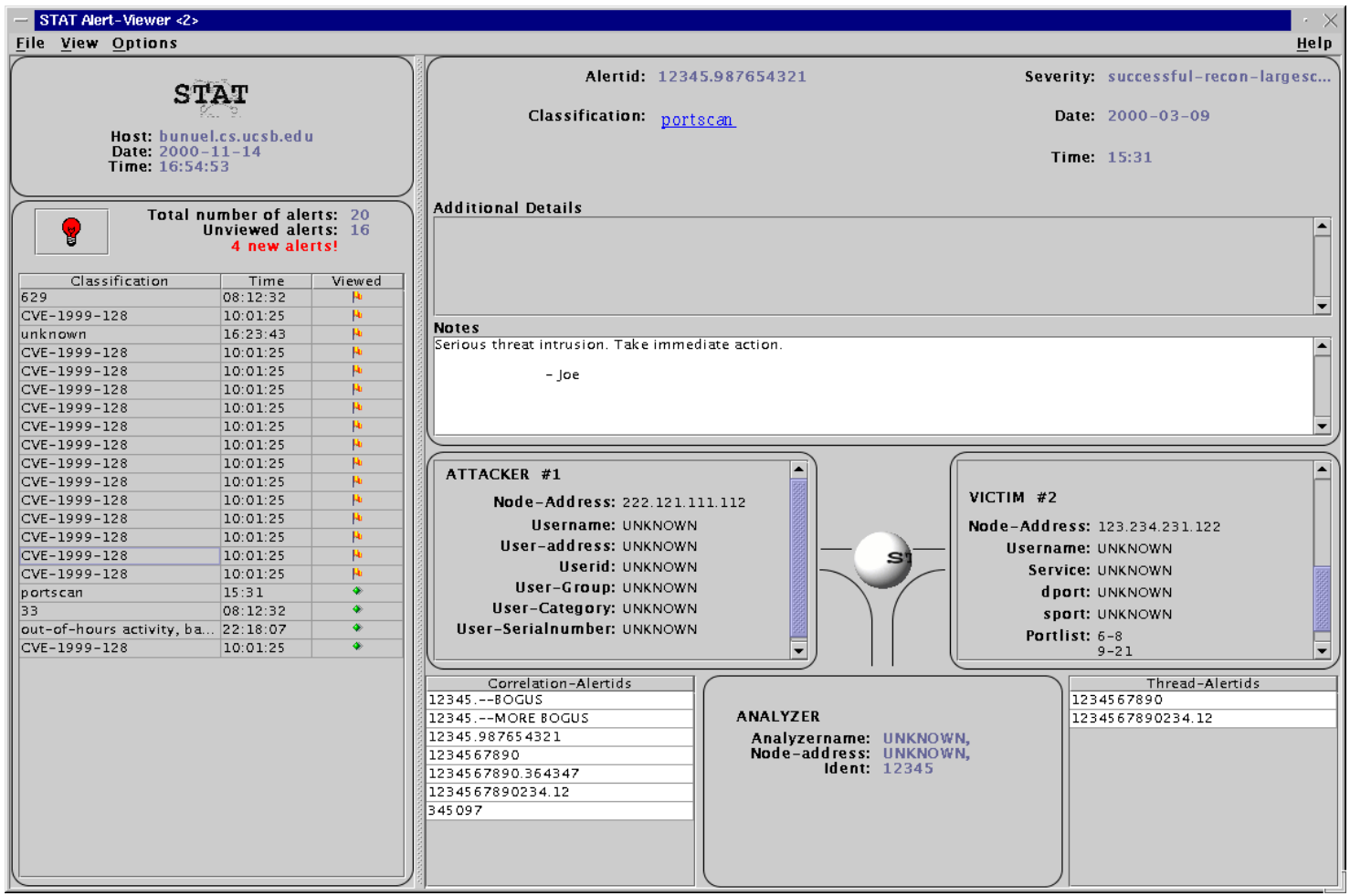
- MetaSTAT Configurator manages sensors
  - Database of available modules and corresponding dependencies
  - Database of current sensor configurations
  - Allows the security officer to submit reconfiguration requests
  - Checks for the meaningfulness of reconfiguration
- MetaSTAT Collector component aggregates sensor alerts in a centralized database to support analysis and correlation

STAT

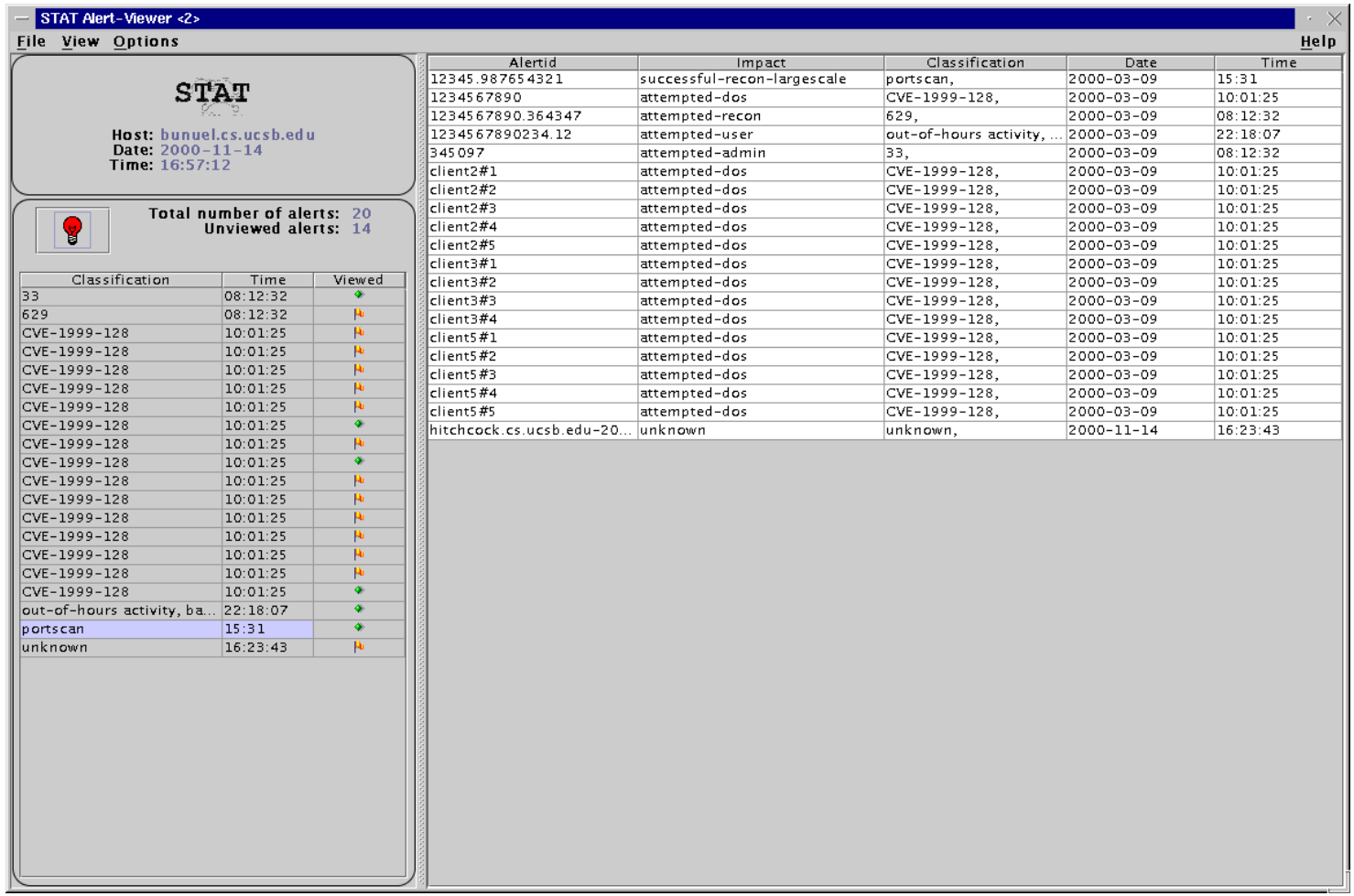
MetaSTAT



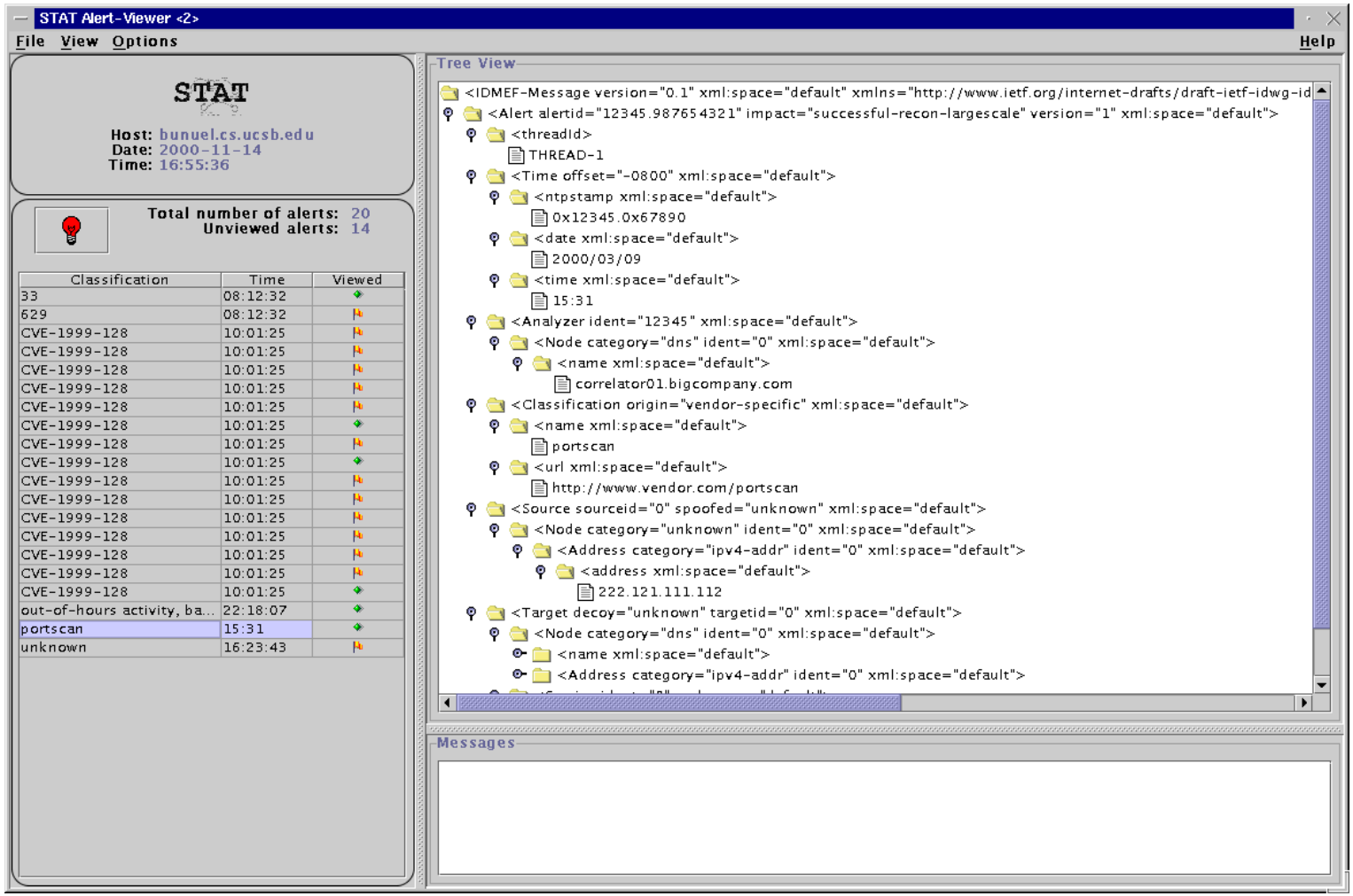
MetaSTAT: Main View

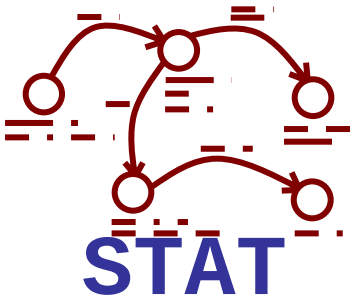


MetaSTAT: Table View



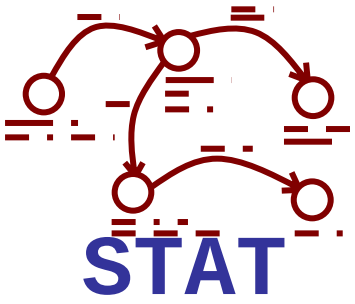
MetaSTAT: Tree View





Module Database

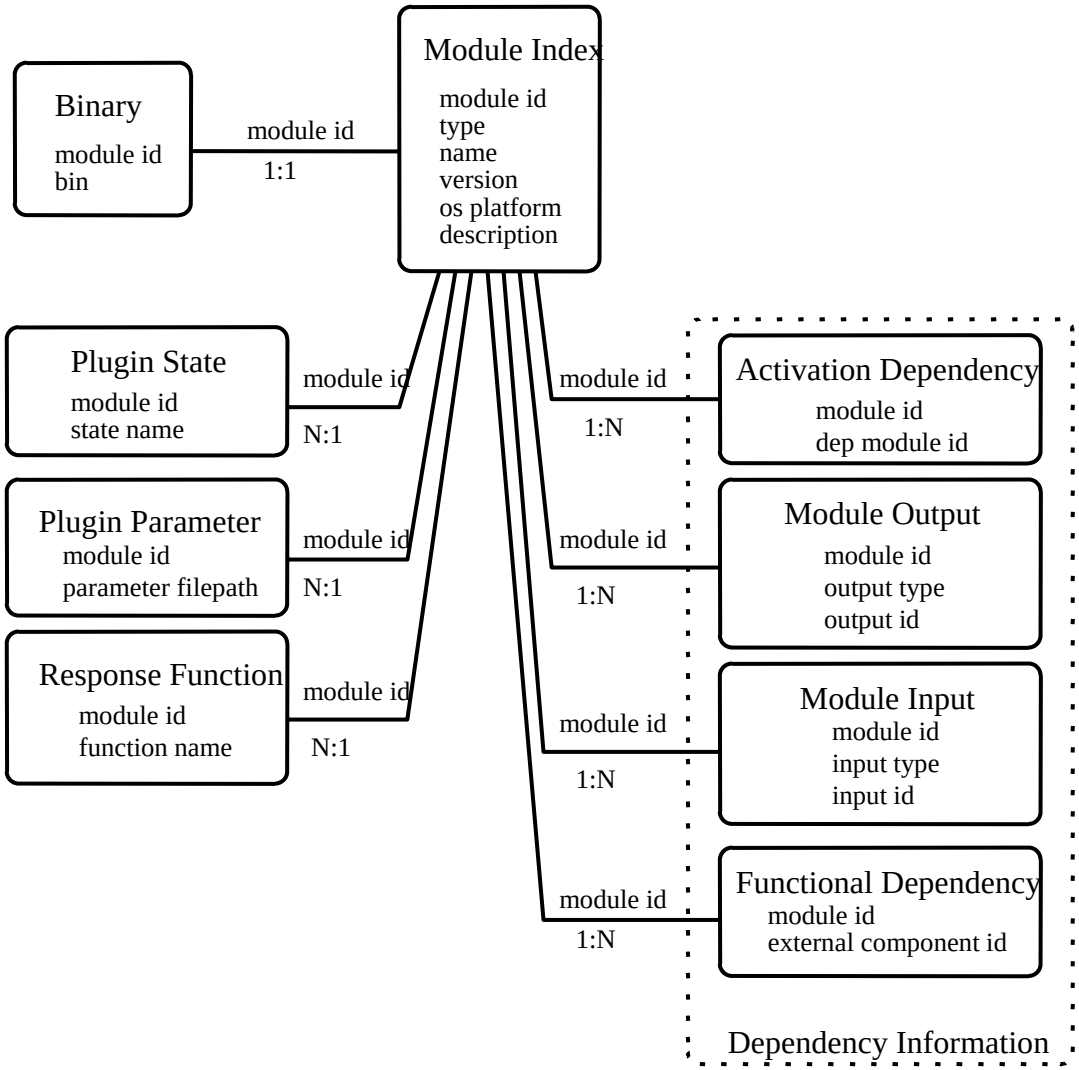
- Models and stores the information about
  - The available *modules* (Language Extensions, Event Providers, Attack Scenarios, and Responses)
  - A number of *external components* (e.g., a specific auditing facility)
- Models and stores the dependencies between modules and components
  - *Activation dependencies:* Module A needs module B in order to be loaded and activated
  - *Functional dependencies*: Module A needs module B in order to produce meaningful results or any results at all

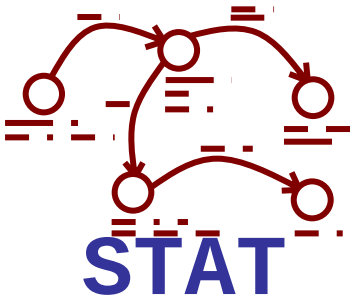


Module Management

- Each Module may be
  - Installed
  - Loaded
  - Activated
- A STAT sensor configuration is uniquely defined by a set of installed/activated modules and available external components
- A configuration is *valid* if all the activation dependencies are satisfied
- A configuration is *meaningful* if it is valid and all the functional dependencies are also satisfied

Module Database Schema

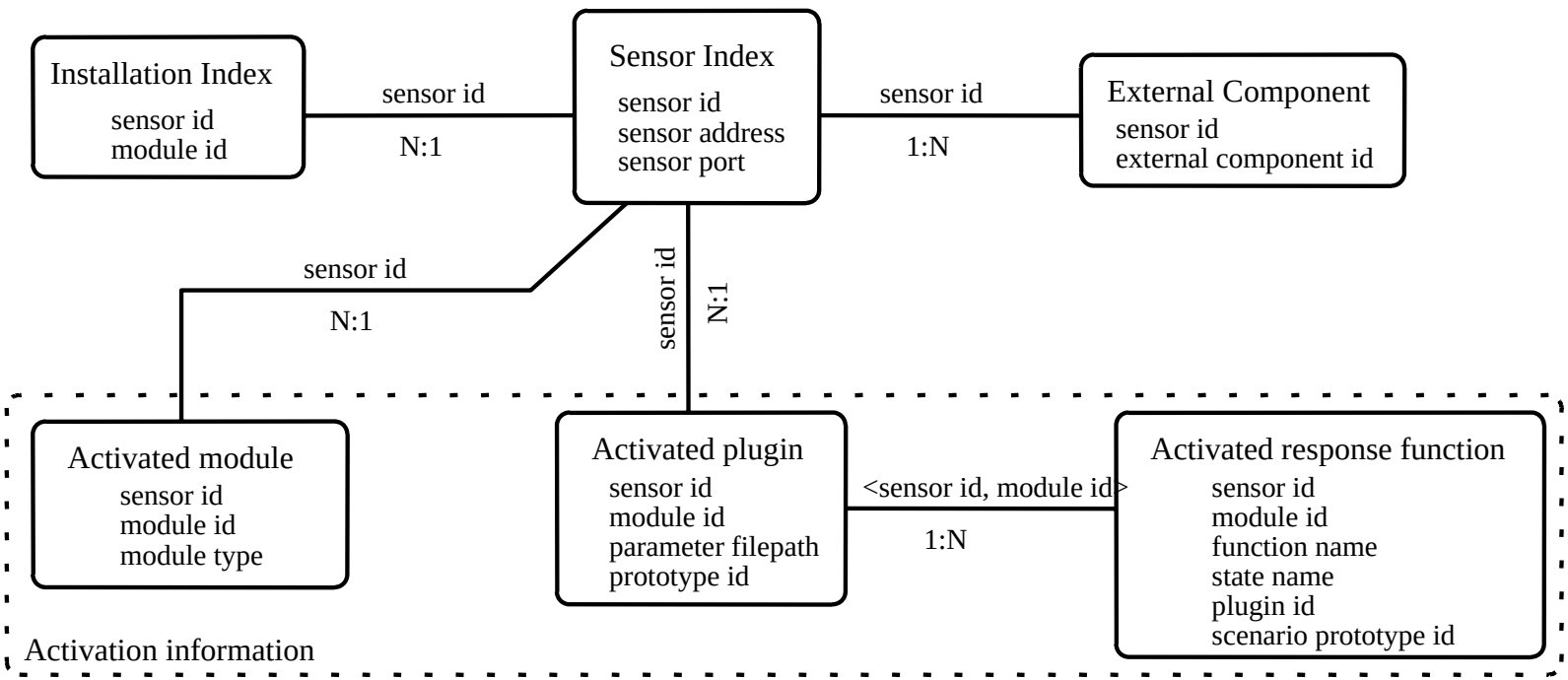


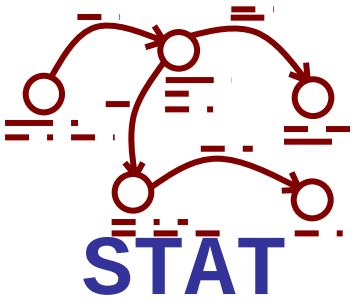


Sensor Database

- Models and stores information about the current configuration of a Web of Sensors
  - Installed modules (at each STAT Proxy site)
  - Loaded/Activated modules (in each STAT Sensor)
  - Available external components (at each host)

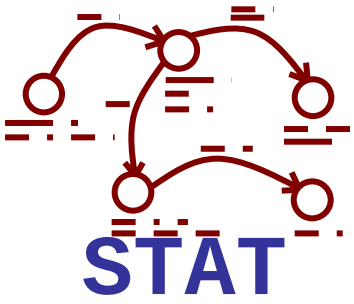
Sensor Database





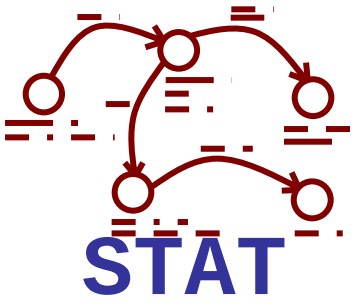
MetaSTAT Configurator

- Intrusion Detection Administrator (IDA) requires high-level reconfiguration
- The MetaSTAT Configurator
  - Determines the required sensor configuration by examining the Module Database
  - Determines which modules are already available using the Sensor Database
  - Determines the steps that are necessary to complete the reconfiguration
- The MetaSTAT Controller sends the appropriate control messages
- STAT Proxies perform the installation
- STAT Sensors reconfigure accordingly

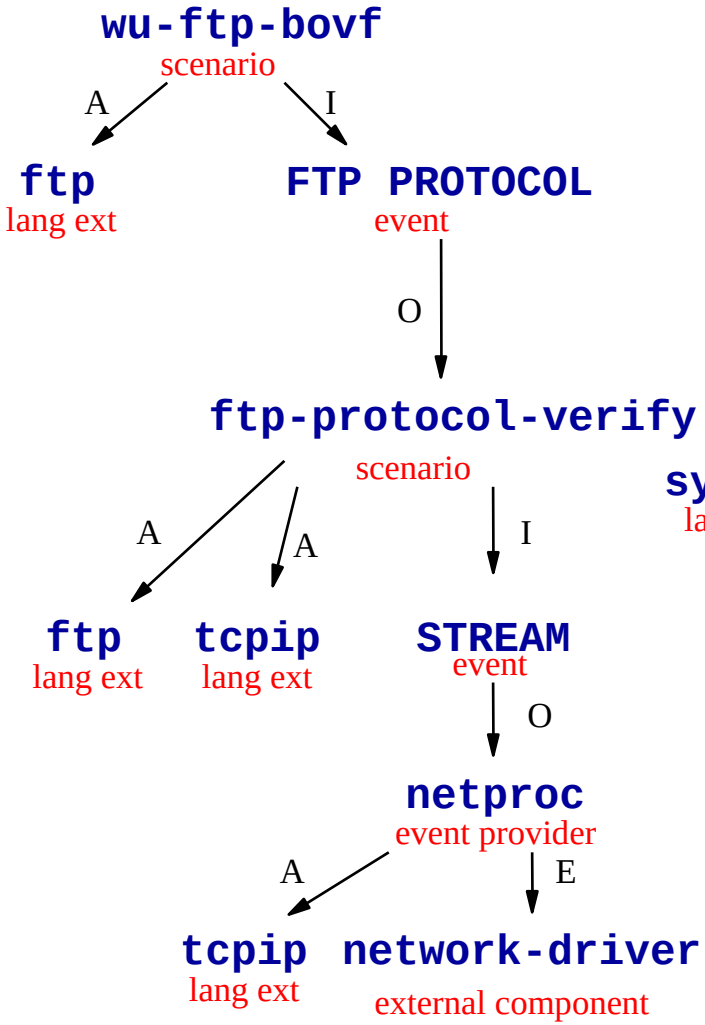


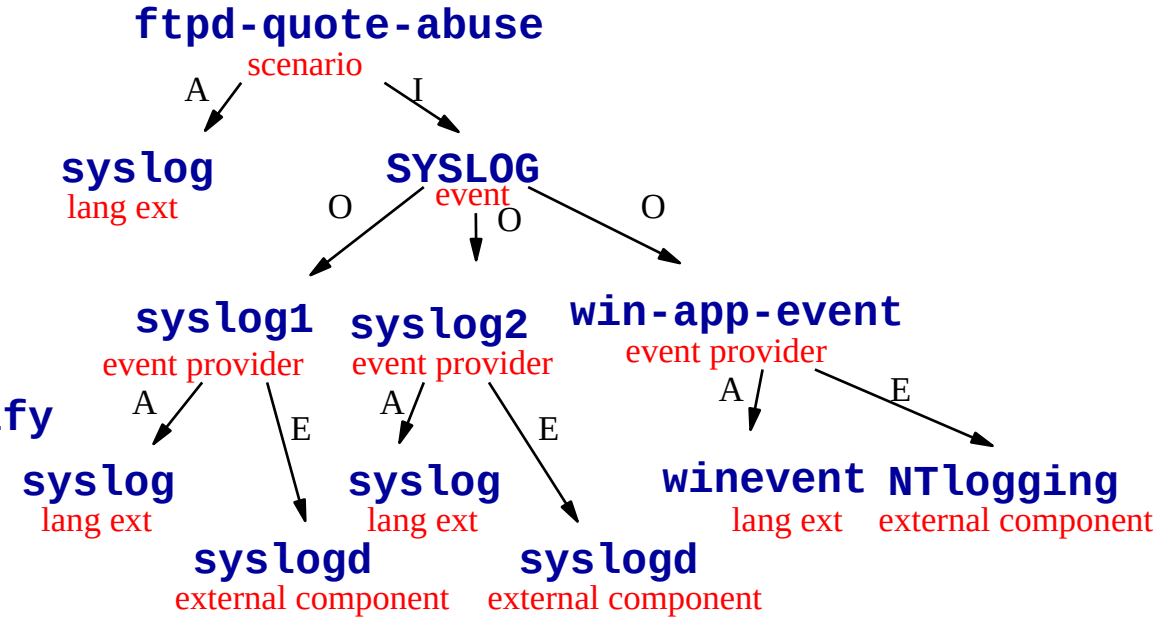
Example

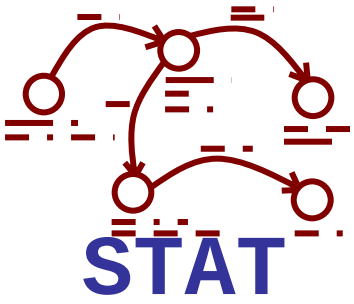
- Intrusion Detection Administrator (IDA) wants to deploy FTP monitoring scenarios
- The Module Database is searched for suitable modules
- A subset is selected
- The Module Database is examined for possible activation dependencies
- The Module Database is searched for possible functional dependencies
- Results trigger a new series of queries



Dependency Graph



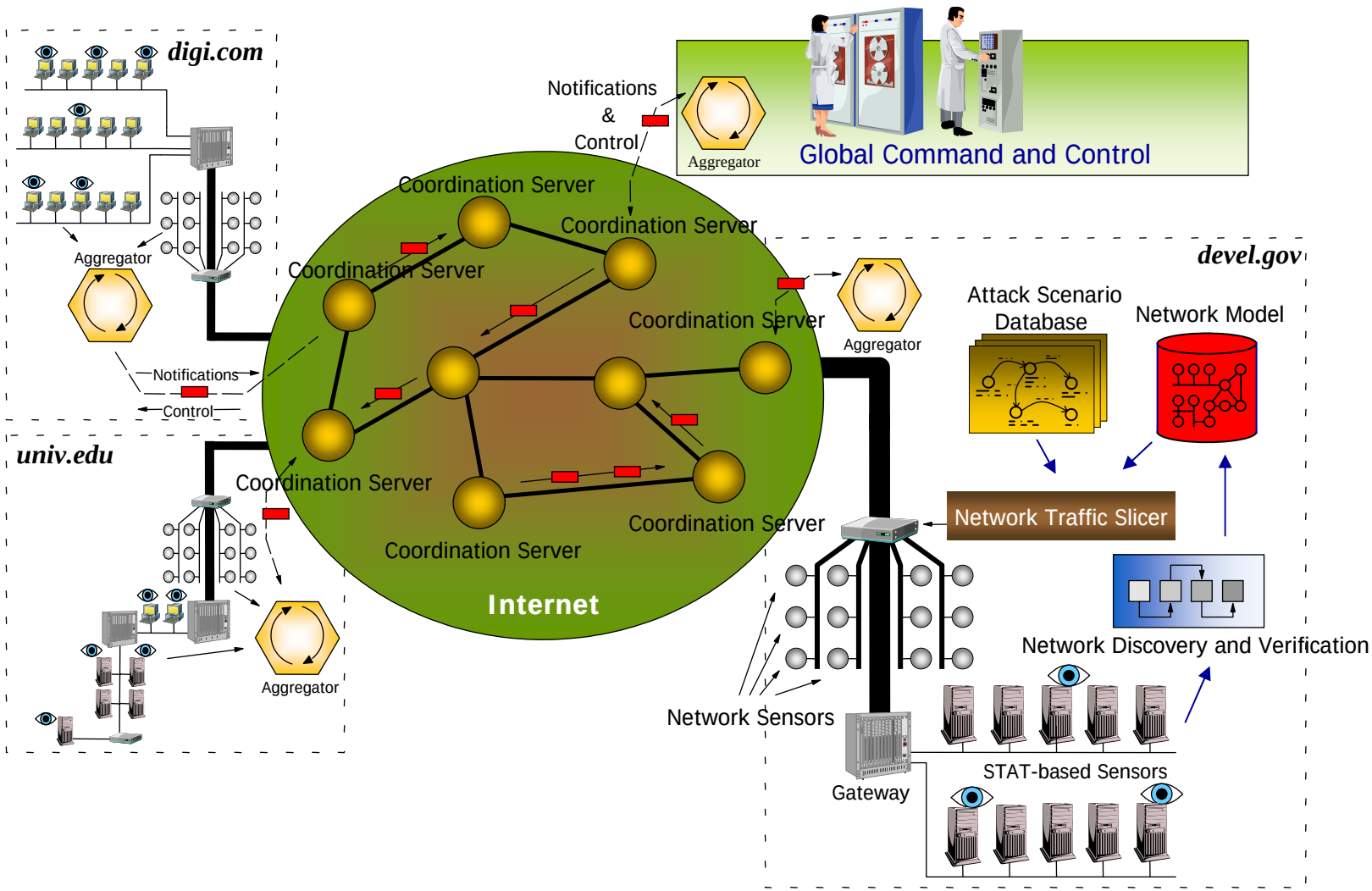


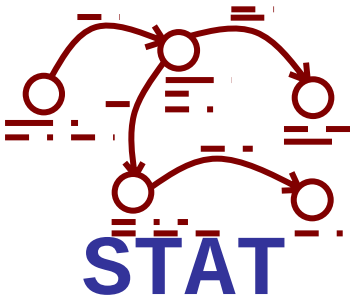


Example

- Configurator determines the complete set of dependencies
- Configurator compares required modules with installed/activated modules as stored in the Sensor Database
- Configurator compiles a *deployment plan*
- Plan passed to the Controller
- Controller ships messages to the Proxies
- Proxies perform installations and forward loading/activation messages to the sensors
- Detection begins...
- Possible custom responses are shipped/installed/activated

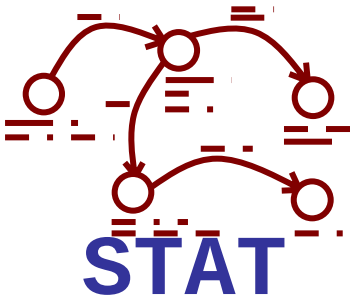
Hi-DRA Architecture





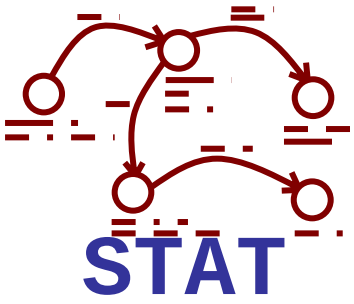
Advantages of the Approach

- Fast development of intrusion detection sensor for different platforms/domains
- Highly customizable
- Dynamic re-configurability
- Support for the management of a very large number of sensors
- Separation of analysis mechanisms from domain-dependent elements and response functionality
- Modules can be reused across sensors



Advantages of the Approach

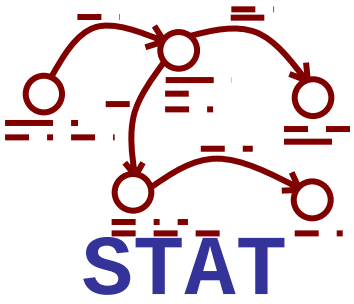
- Multiple Language Extensions and Event Providers can be used within the same sensor
- Responses can be associated with intermediate steps in attack scenarios
- Support for alert collection and distribution
- Third-party tools can be easily integrated through STAT Proxies



People Involved

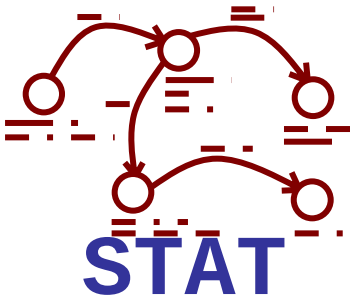
- Richard Kemmerer
- Giovanni Vigna
- Steve Eckmann
- William Robertson
- Fredrik Valeur
- Jingyu Zhou

- Per Blix, Jacob Copenhaver, Marco Cova, Chris Kruegel, Darren Mutz, Rahul Nirmal, Siva Sankaridurg, Tirthendra Sanyal, Vishal Kher, Sunita Verma



Papers

- “NetSTAT a Network-based Intrusion Detection Approach,” 14th Annual Computer Applications Conference, Dec. 1998
- “NetSTAT a Network-based Intrusion Detection System,” Journal of Computer Security, Vol. 7, No. 1, 1999
- “The STAT Tool Suite,” Discex 2000 , Jan. 2000
- “Attack Languages,” Third Info. Surv. Workshop, Nov. 2000
- “STATL: An Attack Language for State-based Intrusion Detection,” Intrusion Detection and Prevention Workshop, Nov. 2000
- “Designing a Web of Highly-Configurable Intrusion Detection Sensors,” RAID01, Oct. 2001



Papers

- “Automated Translation Between Attack Languages,” RAID01, Oct. 2001
- “Intrusion Detection”, IEEE Security and Privacy Magazine, April 2002
- “Stateful Intrusion Detection for High-Speed Networks”, IEEE Symposium on Security and Privacy, May 2002
- “STATL: An Attack Language for State-based Intrusion Detection,” Journal of Computer Security, 2002
- “Designing a Family of Intrusion Detection Sensor,” ESEC 2003, Oct. 2003
- “A Stateful Intrusion Detection System for World-Wide Web Servers,” ACSAC 14, Dec. 2003